# Chip simulation used to run automotive software on PC

Jakob Mauss
QTronic GmbH, Alt-Moabit 92, 10559 Berlin, Germany

***Abstract* - Simulation has great potential to improve the development process for automotive ECUs. Simulation helps to move development tasks to PC, where they often can be performed faster, cheaper or better. However, development tasks such as calibration and test are typically performed by an OEM, while the ECU code is owned by the supplier of the ECU. Therefore, the OEM is often unable to set up a simulation for PC based on the original C code of the ECU. In this paper, we show how to overcome this problem. The key idea is to run automotive software on PC by emulating the target processor of the ECU, which enables simulation of automotive software on PC without accessing the corresponding C code. To be of practical use for automotive development, such a 'virtual ECU' must provide interfaces to automotive standards and tools used for measurement, calibration, and closed-loop simulation. The lack of such automotive interfaces is the main reason that chip simulators, although on the market since decades, have not been used in this domain so far. The paper shows how to bridge the gap by complementing chip simulation with additional services needed to simulate ECU behaviour on the level of detail seen by function developers, calibration engineers or the driver of a car. To demonstrate the usefulness of the approach, we also report how ECU simulation as described here and implemented by our virtual ECU tool Silver is currently used by car makers to develop engine controllers.**

## 1. Introduction

### 1.1 Development of automotive software

ECUs for automotive powertrains (engines, transmissions, hybrid drivetrains) are jointly developed by

- suppliers of ECU hardware
- suppliers of basic software, such as the real-time operating system RTOS and device drivers
- an OEM responsible for the integration of the ECU into a vehicle
- engineering service providers contracted by the OEM and its suppliers

On the OEM side,  about 50% of the engineering effort for an engine or transmission controller is devoted to tuning of the software parameters that determine e.g. fuel economy or emissions of the vehicle. About 40% or all development effort in this domain is spent on test-related activities. A typical development project for a new engine or transmission generates at least 10 releases of the control software, and takes 100 person years spent within 3 years from first prototype to start of production.

The typical tooling and process for development is ahown in Fig. 1: The supplier of the ECU uses a model-based tool chain such as Ascet (ETAS), MATALB/Simulink (MathWorks) and/or TargetLink (dSPACE) to develop a model of the controller and to generate C code from that. Often, the OEM supplies selected software modules as well. The resulting C code is compiled for the target processor of the ECU and the resulting binary is delivered to the OEM, together with

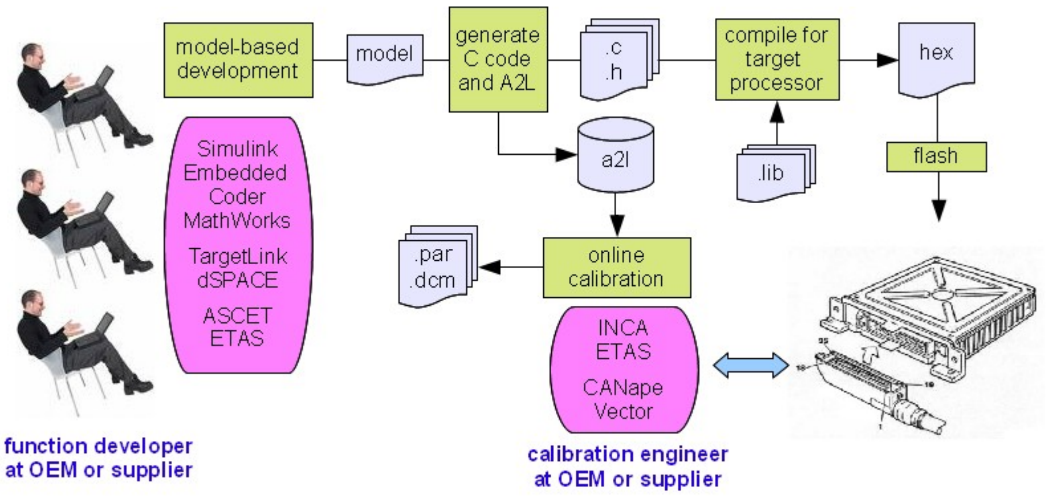


*Fig 1: development process for automotive ECUs*

descriptions such as a map file (list of functions and variable names and their corresponding adresses) and an ASAP2/a2l description of the entire ECU. The ASAP2/a2l file describes all tunable parameters of the ECU and all variables that can be measured at runtime. Each description includes address in ECU memory, datatype (e. g. "*10 x 16 array of signed 16 bit integer*"), and conversion rules (e.g. "*divide the raw integer by 10 to get pressure in bar*"). ASAP2/a2l is a standard format [11] maintained by ASAM e.V. and supported by tools used for online calibration and measurement, such as CANape (Vector), INCA (ETAS) or CalDesk (dSPACE). A major task on the OEM side is then to test the ECU and to tune the thousands of software parameters of the ECU to optimize overall performance of the vehicle. Calibration and test is typically perfomed in the vehicle on the road, or on engine and transmission test rigs, roller dynamometer test benches, or hardware in the loop (HiL) test rigs. In all cases, at least the ECU that runs the control software is physically present, while the environment might be partially simulated.

### 2.1 Virtual ECUs and the need to integrate chip simulators

In this paper, we consider the case that also the ECU itself is simulated. Such a 'virtual ECU' can be used to move development tasks to PC, where they often can be performed faster, cheaper or better. For example, if the entire system (vehicle and ECU) is simulated, there is no real-time constraint any more. Selected ECU functions can then e.g. be simulated 30 times faster than real time, which enables the use of mathematical optimization to compute optimal engine maps: An experiment that takes a month to complete on a engine test rig can then be done in a day on a simple PC. Such a simulation can also be halted ("freeze time") and stepped, to support the debugging and test process, or coupled with a detailed simulation model (running much slower than real-time), e. g. for closed-loop analysis of the combustion process. A virtual ECU can also be used to partially shift the test process from expensive and slow HiLs to cheap and fast PCs. There are in principal three options to set up an ECU simulation on PC

- Re-host the C code. Compile the C code of the ECU for execution on Windows PC. Obviously, this requires access to the C code.
- Reverse engineering. Reimplement selected ECU functions of interest using a tool for modelling and simulation such as MATLAB/Simulink. This does not require access to the C code, but is both error prone and time consuming. Besides, legal constraints (e. g. license conditions) might prohibit this option.
- Chip simulation. Run the native ECU code on PC by emulating the ECU processor. This requires neither access to the C code nor reverse engineering.

Despite the potential to speed up development, virtual ECUs are still rarely used by automotive OEMs, because of difficulties to effectively set up a virtual ECU for simulation on PC. Often, re-hosting is not an option, due to missing C code. Reverse engineering fails due to legal concerns or due to bad cost-benefit ratio. Chip simulation is typically not an option because chip simulators available on the market lack required interfaces to automotive development tools. To illustrate the last point: A typical engine controller contains about 50.000 scalar variables that can be measured during operation, and approximately the same number of tunable parameters, with hundreds of scaling rules needed to convert raw integers to meaningful values, such as angular speed or pressure. A chip simulator alone does not help here. Additional services are needed to interface running ECU code e. g. with a vehicle simulation model.

The main contribution of this paper is to demonstrate how to effectively use chip simulation to support automotive ECU development in scenarios where neither C code nor detailed documentation of the ECU hardware is available. Our solution, implemented by the virtual ECU tool Silver, combines the following elements

- A fast instruction-accurate chip simulator that runs native ECU code by emulating the ECU processor
- A technique that uses the standardized ASAP2/a2l description [11] of the ECU to bypass on-chip devices such as CAN controllers, AD/DA converters and all features of the ECU board. This way, an ECU simulation can be set up quickly, without requiring knowledge about the ECU board or the specific on-chip devices of the target processor.
- Interfaces to simulation tools, used to run ECU functions in closed-loop with a vehicle model. This includes automatic conversion of interface variables to and from raw integers to physical quantities.
- Interface to enable online measurement and tuning of the virtual ECU using standard protocol. This way, existing procedures for measurement and calibration continue to work also for virtual ECUs.

The remainder of this paper is structured as follows. In section 2, we describe our approach to ECU modelling and simulation on user level, as well as the technical realisation and performance of our chip simulator. Section 3 reports various use cases in real automotive development projects. In Section 4, we describe the limitations of our approach and in Section 5, we discuss related work.

## 2. Simulation of ECUs with Silver 3.1

Silver [1,2,10] is a tool used to build and run virtual ECUs on Windows PC, offering support for re-hosting given C code on PC, as well as building virtual ECUs from native binaries via chip simulation. Silver is mostly used to develop automotive power trains.

To virtualize ECUs with Silver via chip simulation, the

ECU is conceptually decomposed into layers as shown in Fig. 2. The ECU hardware provides electrical interfaces to sensors and actuators and resources to run a real-time operating system. The RTOS runs tasks, either initially, periodically or event triggered. A task is a function and may call additional functions. At run time, functions read and write variables from certain locations in the RAM, and read application data (maps, curves, scalar constants) from ROM. The application data needs to be tuned by calibration engineers to optimize system performance.

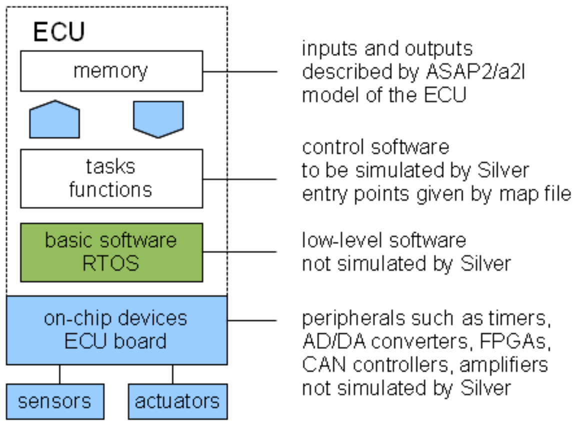


*Fig 2: Layers of an ECU*

The objective of virtualization as presented here is to run the tasks and functions on PC, such that the system performance can be assessed or optimized via simulation, either in closed-loop with a vehicle model, or open loop driven by synthetic or measured inputs. In our context, system performance is measured w.r.t aspects such as shift quality, fuel economy, or engine emissions, i.e. aspects that a calibration engineer can control. We are not interested here in capturing low-level performance, such as energy consumption of the CPU, clock cycles spent on a task, bus load, pipeline and cache performance and the like. The focus on system-level helps to achieve a very good cost-benefit ratio of our virtualization approach: A simulation can be set up quickly and requires only little information about the ECU and the chip, such as just the 19 lines shown in Fig. 3. The latter point is crucial, because more detailed information may not be available in the application scenario sketched in the introduction.

Silver 3.1 supports two alternative chip families: Infineon TriCore and Freescale PowerPC.  Many automotive controllers are based on these processors families, in particular in the power train domain. Examples are engine controllers of the MED and EDC family by Bosch, and transmission and engine controllers by Continental, Delphi and Magneti-Marelli.

An ECU simulation is set up as follows: The user has to write a specification file (similar to the OIL file used to configure OSEK) to specify, which tasks of the ECU to simulate. Silver automatically turns such a spec file into an executable Silver module (a Windows DLL) or a MATLAB SFunction.

A typical spec file looks as shown in Fig. 3. The hash # character starts a comment, which is ignored by Silver. The spec file first lists the required files (line 2-5). The map file is optional. If a map file is given, the spec file may use symbolic names for functions (such as ABCDE_20ms). Otherwise, addresses (such as 0x80081cde) must be used.

Lines 9 - 13 lists the functions to run, and specifies when and in which order to run these functions. Silver uses this to emulate the RTOS. For event triggered tasks, Silver offers two alternative event models. Line 11 shows a function that is executed *n* times at each base clock tick, where *n* is the value of the input variable trigger at the tick. Typically, *n* is 0 or 1 during simulation. Higher values occur only, when more than one trigger event

```
01 # specification of sfunction or Silver module
02 hex_file(m12345.hex, PowerPC)
03 a2l_file(m12345.a2l)                 # ASAP2/a2l model of the ECU
04 map_file(m12345.map)                 # a TASKING or GNU map file
05 chip_config(BASE_CLOCK, 10)          # base clock of the RTOS in ms
06 chip_config(USER_STACK, 0xd0001000)  # location of stack
07
08 # functions to be simulated, in order of execution
09 task_initial(ABCDE_ini)              # run initially
10 task_initial(ABCDE_inisyn)           # run initially
11 task_triggered(ABCDE_syn, trigger)   # run event triggered
12 task_periodic(ABCDE_20ms, 20, 0)     # run every 20 ms, with offset 0
13 task_periodic(ABCDE_200ms, 200, 0)   # run every 200 ms, with offset 0
14
15 # interface of the generated sfunction or Silver module
16 input(nmot)
17 output(target_cur_mv1)
18 output(target_cur_mv2)
19 parameter(mv_curve)
```

*Fig 3: Specification file for a ECU Simulation in Silver*

occurs between two clock ticks. Silver also offers a more accurate event model, that allows execution of an event triggered task at exact event time, not just at clock ticks.

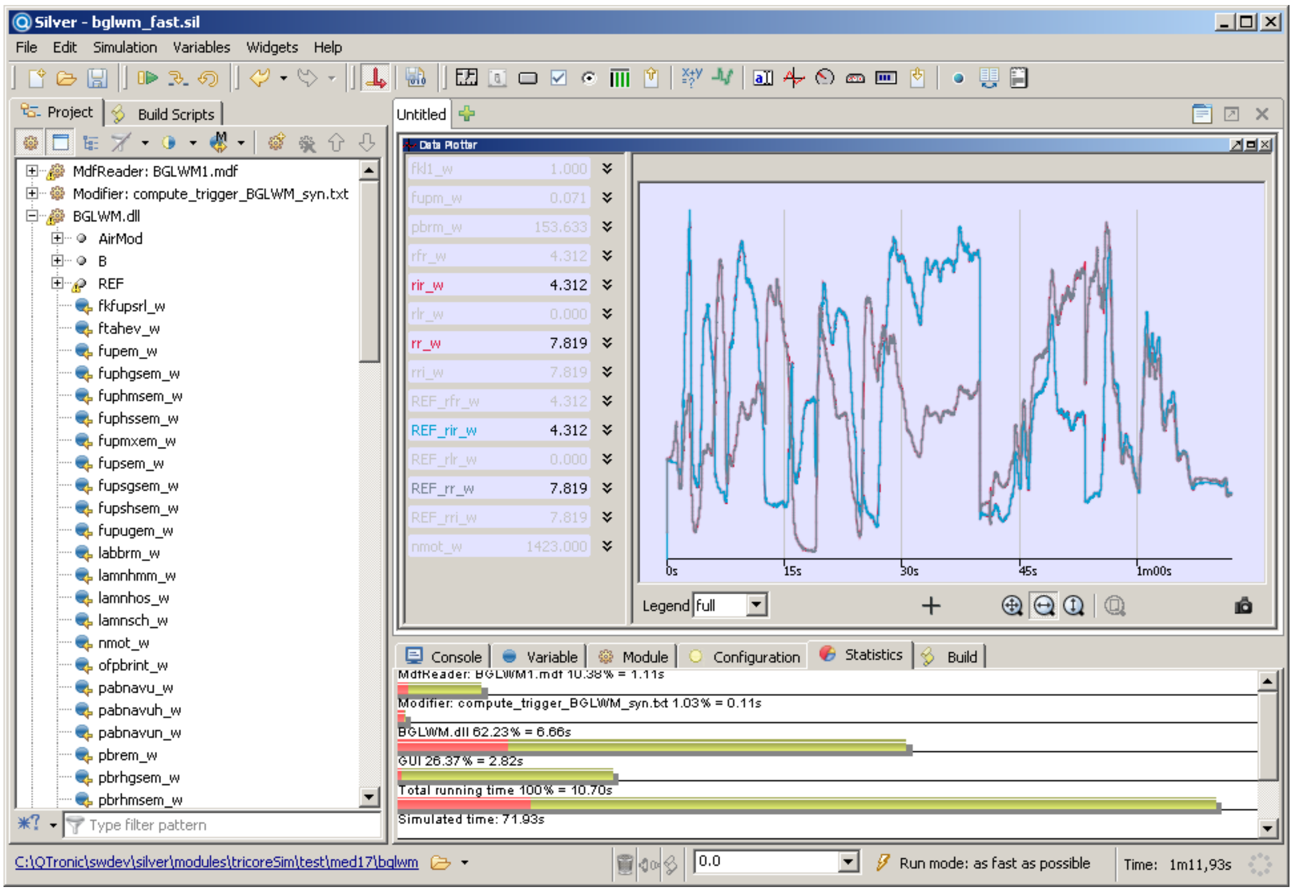
Finally, lines 16-19 define the inputs, outputs and parameters of the generated module or SFunction, using names defined in the a2l file.

In addition, Silver offers means to specify

- properties of the XCP emulation, if any, to support online calibration and measurement using tools such as INCA or CANape
- data sections to be included into the generated Silver module or SFunction. This way, initial loading of the hex file into simulated memory can be avoided, to speed up simulation.
- memory areas to be copied to other (faster) memory by the start-up code
- functions to be replaced by other functions. This way, a function called to access sensors or actuators can be replaced by a function that directly accesses a plant model or measured values instead. Another use case is virtual prototyping when developing of new ECU functions.
- logging options, e.g. to track program execution and memory access during simulation

The spec file used to port selected parts of a hex file to PC might contain bugs. To locate bugs, Silver integrates a debugger to be used whenever a simulation does not perform as expected, i.e. differs from measured behaviour. This allows to step though the program code, one instruction per step, allowing a user to inspect register content before and after execution of an instruction. It is also possible to set code and data breakpoints, for example to pause a simulation whenever a certain variable is accessed.

ECU simulations are typically validated against behaviour measured in the vehicle or on a test rig. This is shown in Fig. 4, where a measurement file is used to drive a simulated function of the MED17 ECU, a gasoline engine controller by Bosch. Function outputs (red) and measured outputs (blue) are plotted for two variables, and show only small deviations. The simulation itself is bit-accurate, i.e. the simulator performs exactly the same computations as the real target. Small deviation between measured and simulated behaviour are however possible, either  because tasks run and slightly different times, or due to different sampling rated used for measurement. For the application described in section 3, such deviations are typically irrelevant.


Silver - bglwm_fast.sil

File Edit Simulation Variables Widgets Help

Project | Build Scripts

- MdfReader: BGLWM1.mdf
- Modifier: compute_trigger_BGLWM_syn.txt
- BGLWM.dll
  - AirMod
  - B
  - REF
  - fkfupsrl_w
  - ftahev_w
  - fupem_w
  - fuphgsem_w
  - fuphmsem_w
  - fuphssem_w
  - fupmxem_w
  - fupsem_w
  - fupsgsem_w
  - fupshsem_w
  - fupugem_w
  - labbrm_w
  - lamnhmm_w
  - lamnhos_w
  - lamnsch_w
  - nmot_w
  - ofpbrint_w
  - pabnavu_w
  - pabnavuh_w
  - pabnavun_w
  - pbrem_w
  - pbrhgsem_w
  - pbrhmsem_w

Type filter pattern

Untitled

Data Plotter

| | |
|---|---|
| fkl1_w | 1.000 |
| fupm_w | 0.071 |
| pbrm_w | 153.633 |
| rfr_w | 4.312 |
| rir_w | 4.312 |
| rlr_w | 0.000 |
| rr_w | 7.819 |
| rri_w | 7.819 |
| REF_rfr_w | 4.312 |
| REF_rir_w | 4.312 |
| REF_rlr_w | 0.000 |
| REF_rr_w | 7.819 |
| REF_rri_w | 7.819 |
| nmot_w | 1423.000 |

Legend full

Console | Variable | Module | Configuration | Statistics | Build

```
MdfReader: BGLWM1.mdf 10.38% = 1.11s
Modifier: compute_trigger_BGLWM_syn.txt 1.03% = 0.11s
BGLWM.dll 62.23% = 6.66s
GUI 26.37% = 2.82s
Total running time 100% = 10.70s
Simulated time: 71.93s
```

C:\QTronic\swdev\silver\modules\tricoreSim\test\med17\bglwm

0.0

Run mode: as fast as possible

Time: 1m11,93s

*Fig 4: ECU function (MED17 engine controller by Bosch) running in Silver*

The chip simulator used by Silver is an instruction accurate (as opposed to cycle accurate) instruction set simulator, that has been optimized for high execution speed. The simulator uses binary translation to map the instruction set of the target processor (TriCore or PowerPC at present) to the instruction set of the host processor (typically Intel x86). Compilation is not done 'just in time' here [12], i.e. at run time, but mostly at compile time, based on automatic identification of reachable code. Code that has been missed by that compile-time pre-fetch (because it was not recognized as reachable) is processed by an instruction interpreter at runtime. This way, speed penalties for compilation at runtime are mostly avoided. Missed code is reported during simulation and can be included by rebuilding the virtual ECU, if the expected speed-up is worth that extra effort.

In order to measure the execution speed of Silver's chip simulator, we have ported a complex ECU function (from MED17 with TC1797) implemented by 5 different C functions that run initially, every 10 and 200 ms, and synchronous to the crankshaft. The spec file is very similar to the one shown in Fig 3. The function has 114 scalar inputs, 102 scalar outputs and 108 parameters (characteristics), many of them axes and maps. We have then measured all inputs and outputs of the function on an engine test rig for a scenario of 3.5 minutes and used the resulting measurement (a mdf/dat file) to drive simulations in Silver, using a Silver module generated from the spec file. All reachable code had been detected and compiled. Therefore, slow interpreter mode was not used at all during simulation. Each simulation (shown in Fig. 4) executed 380.205256 million instructions and has been repeated 5 times on a PC with Intel i7 processor at 2.67 GHz and 4 GB RAM under Windows 7. The average execution time found this way was 62 MIPS. The target processor TC1797 operates at about 180 MHz, which corresponds to about 270 MIPS for typical embedded applications. One might hence think, that simulation runs by a factor of 4.35 slower on PC than on the real target. This is however not the case. The real ECU is constrained to real time. That means, running a scenario of 210 seconds on the real ECU takes exactly 210 seconds. Silver has no such constraint: running a 210 second scenario on PC using the 62 MIPS simulator takes just 5.9 sec, which is 35 times faster than real time. We did a similar experiment with Silver's chip simulator for Freescale PowerPC and found that Silver runs automotive applications compiled for this chip family with about 200 MIPS on average, using the same Windows PC as above.

Silver can also turn a spec file into a SFunction, i.e. a mexw32 file that runs in MATLAB/Simulink. This is particularly interesting when using chip simulation to support automated optimization of parameters, because many optimization tools are implemented on top of MATLAB/Simulink. The generated SFunction accepts all characteristics listed in the spec file as SFunction parameters. This makes it easy to connect the generated SFunction with an optimization procedure. For example, the SFunction can be called with workspace variables that are then automatically varied by the optimization procedure between SFunction calls. The performance of a generated SFunction is again about 40 MIPS.

## 3. Applications

Calibration engineers use tools such as INCA (ETAS) or CANape (Vector) to connect to a running ECU and to online tune its parameters during operation. Online calibration is based on the XCP protocol. The simulation models generated by chip simulation implement that protocol. Therefore, INCA and CANape can both be connected to a chip simulation running on PC, to online-tune parameters. This way, calibration tasks can be moved from road and test rigs to highly available and cheap PC platforms, as shown in Fig. 5. When using Silver, the FMI standard [4] can be use to import a vehicle model from other simulation environments in order to set up a closed-loop calibration environment. Many calibration tasks can however be performed using much simpler open-loop setups, i.e. without getting a vehicle model into the simulation loop.

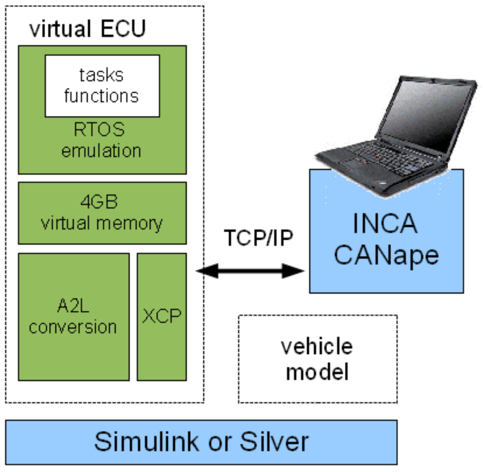


*Fig 5: On-line calibration of a virtual ECU running on a PC*

During development of an engine controller, a developer might want to replace a certain function of the ECU by its own version of that function, bypassing the original function. For real ECUs, this can be done with tools such as EHOOKS (ETAS) or No-Hooks (ATI). These tools manipulate the original hex file, such that the bypassed function is not executed any more, but just calls the new function instead. The placing of bypass hooks by direct manipulation of the hex file is a mighty but error-prone tool. Sometimes a hooked function is not called at all or only some variables are overwritten and some not. Normally, such errors are only detected after the manipulated hex-file was flashed on the ECU and then run on the test bench or in a car. With the possibility of instruction accurate simulation of the patched hex file, we can detect these errors much faster and without any risk to car or engine.

A third application of chip simulation is the numerical optimization of engine parameters. [6, 7] reports how to combine chip simulation as described above with various procedures for numerical optimization to compute optimal values for certain engine control parameters. These computations require an accurate and fast model of the engine function of interest. In the past, calibration engineers had to use hand-coded models of ECU functions, developed e.g. with MATLAB/Simulink. This is time consuming and error prone. Tedious reverse engineering can now be replaced by a faster and cheaper process based on SFunctions generated automatically by Silver from a given hex file. Fig. 6 shows an example from [6]. The SFunctions generated this way proofed to run as fast as their hand coded counterparts.

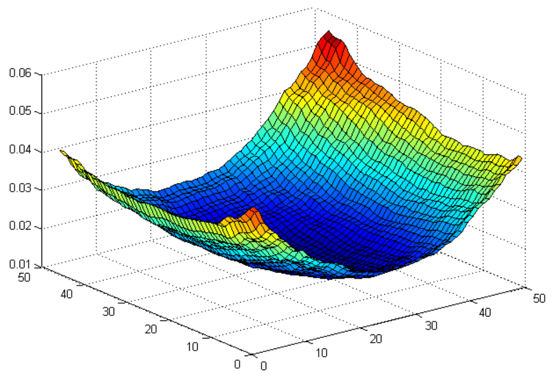


*Fig 6: Goal function for optimization with Simulink, implemented via chip simulation*

## 4. Limitations and future work

The virtual ECU generated by Silver performs exactly the same computations on PC, as on the real target, since the effect of every machine instruction on memory and chip registers is exactly simulated on PC. However:

- simulation is just instruction accurate, not cycle accurate. This means, the simulation on PC cannot be used to exactly predict execution time on the real target. For example, pipeline effects or different access times to memory (e. g. fast on-chip RAM versus external RAM) are not modelled.
- conceptually, a simulated task executes infinitely fast. This means that the emulated RTOS never interrupts a task. In particular, parallel execution of code on multi-core processors is not simulated as such. Related effects cannot be analysed using the generated model.
- Silicon bugs are not simulated. If a compiler for the real target does not work around a silicon bug correctly, this is likely to be invisible in the simulation: simulated behaviour and behaviour on the real target might differ in such cases.
- On chip devices such as CAN controllers, AD/DA converters, timers, the PowerPC eTPU and the TriCore PCP co-processor or are not simulated.

The above limitations exclude for example the following use cases:

- development and test of basic software, such as device drivers and CAN controllers: not possible with Silver because the on-chip devices that the basic software interacts with are not emulated.
- test whether real-time constrains are met: not possible because conceptually, a task runs infinitely fast on the virtual ECU, i.e. the number of clock cycles spent by a task is not predicted.
- run an entire hex file, including start-up code and RTOS, not just selected tasks. Not possible because some of the tasks will interact with on-chip devices, e. g. to initialize the CAN controller. Since these devices will not respond as expected, this will trigger serious fault reactions, such as ECU reset.

To overcome the above limitations, we would need to complement the chip simulator with models of the on-chip devices, to be implemented e.g. using SystemC [8] and TLM 2.0. Such extensions have to be developed for each target chip, i.e. vary within a chip family. Moreover, we might need to switch to a cycle-accurate simulation model. Technically, this is possible, but hardly justified by the added value, at least for the applications considered in the previous section.

Another limitation, or at least inconvenience, is the following: To set up a specification file for Silver, a user has to know which tasks to run in order to compute given outputs. However, the map file typically contains thousands of function names, and the ASAP2/a2l description of the ECU gives only weak hints for picking the 'right' functions, e. g. via the a2l FUNCTION element: as a heuristic, a function called ABCDE in the al2 file is often implemented by functions called ABCDE_10ms, ABCDE_20ms etc. in the map file. The question arises how to better support the user with selecting the 'right' functions to simulate. We are currently working on automated support for selecting functions, based on symbolic execution of the program code.

## 5. Related work

Instruction set simulators (ISS) similar to the one used by Silver have been around since decades, and we do not attempt to summarize their history here. In short, two types of ISS can be distinguished:

- Cycle accurate ISS model a processor on register transfer level (RTL), i. e. in terms of clocked update of register values, computed as function of other register values. Such ISS are able to predict how many clock cycles an instruction consumes, taking into account the dynamics of instruction pipelines and different access times for registers and memory. See [12] for an example. Cycle accurate ISS are typically used by chip makers for hardware/software performance estimation, optimization, and for functional and timing verification.

- In contrast, instruction accurate ISS sacrifice the ability to generate exact timing predictions to speed up simulation by at least factor 10. The ISS used here is of this type. Such ISS are typically used to build virtual system prototypes (VSP), i. e. simulators of processors or entire ECUs that allow to develop and run application software before real hardware becomes available.

Synopsis is a leading provider of design tools and simulators in the electronic design automation (EDA) domain, covering all scales from transistor level up to register transfer (RTL) and transaction level (TLM). In 2010, Synopsis acquired VaST and CoWare, much smaller companies that had both specialized on VSPs. SystemC [8] is both, a EDA language and a free, open source C++ library that provides an API to simulate concurrent components, typically TLM 2.0 models that represent the peripheral devices of a CPU. Most VSP simulators are based on the SystemC and the TLM 2.0 standard. A notable exception is the Open Virtual Platform OVP by Imperas, which defines an independent exchange format for chip models. OVP supports but is not based on SystemC and TLM 2.0. VSPs relate to Silver as follows:

- In contrast to most VSPs, Silver does not use SystemC or TLM 2.0 as a modeling framework. Silver relies on its ISS alone, without emulating on-chip devices and peripherals, which are by-passed as described in section 2. To treat concurrency, Silver provides a powerful co-simulation framework, based on the FMI standard [4] and on proprietary formats such as MATLAB Sfunctions. This enables Silver to run virtual ECUs in closed-loop with vehicle models imported from simulation systems such as MATLAB/Simulink, Dymola, SimulationX, MapleSim, AMESim, GT-Power, and axisuite.
- In contrast to a VSP, Silver does not virtualize the entire system on a chip (SoC). Hence, Silver cannot run an entire application including start-up code and RTOS. Instead, the user selects all or specific tasks executed by the RTOS, and runs only these in Silver. This has two advantages: First, such simulations are very fast, because Silver runs only a fraction of the code that the real ECU will run. This is key for the application to optimization presented in section 3. Second, setting up such a simulation requires nearly no information about the target ECU, which greatly simplifies setup of such simulations.

Silver addresses the problem of running compiled ECU code on PC without accessing the underlying source code. A technical alternative is to connect a PC to the real ECU based on the standard JTAG debug interface. JTAG is used by many PC-based IDEs to debug applications that run on the target processor. This way, program behaviour can in principle be observed on PC, which could be used as alternative to Silver. However, this approach has several drawbacks

- the real ECU (or at least compatible hardware) is needed, while Silver just requires a binary file
- connecting the ECU to PC via JTAG can be a challenge
- simulation speed as achieved with Silver (e.g. 30 times faster than real time) is difficult to achieve this way, because the ECU must run in real time.

The last point is of particular importance, because mathematical optimization of control code is a major use case here, and to be feasible, high simulation speed is required.

## 6. Conclusion

As demonstrated above, native code of all or selected tasks of an ECU can be executed by the virtual ECU tool Silver on Windows PC, either open-loop driven by measurements or in closed-loop with a vehicle model. This kind of simulation opens new possibilities to move development tasks from road, test rig or HiL to PCs, where they can be processed faster, cheaper or better, without requiring access to the underlying C code. Daimler and IAV currently use this innovative simulation approach to support controls development for diesel and gasoline engines [6, 7]. As reported in [9], Daimler currently maintains about 120 Silver installations to develop engine and transmission controllers for passenger cars.

## References

[1] A. Junghanns, R. Serway, T. Liebezeit, M. Bonin: Building Virtual ECUs Quickly and Economically, ATZ elektronik 03/2012, June 2012. http://qtronic.de/doc/ATZe_2012_en.pdf

[2] H. Brückmann, J. Strenkert, U. Keller, B. Wiesner, A. Junghanns: Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL. International VDI Congress Transmissions in Vehicles 2009, Friedrichshafen, Germany, 30.06.-01-07.2009. http://qtronic.de/doc/DCT_2009.pdf

[3] K. Röpke (ed.): Design of Experiments (DoE) in Engine Development. Expert Verlag, 2013.

[4] T. Blochwitz, M. Otter et. al.: Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. 9th International Modelica Conference, Munich, 2012. http://www.ep.liu.se/ecp/076/017/ecp12076017.pdf

[5] M. Tatar, R. Schaich, T. Breitinger: Automated test of the AMG Speedshift DCT control software. 9th International CTI Symposium Innovative Automotive Transmissions, Berlin, 2010.

http://qtronic.de/doc/TestWeaver_CTI_2010_paper.pdf

[6] M. Simons, M. Feier, J. Mauss: Using Chip Simulation to optimize Engine Control. 7th Conference on Design of Experiments (DoE) in Engine Development, Berlin, 2013. In [3]. http://qtronic.de/doc/doe2013_chip_simulation.pdf

[7] D. Rimmelspacher, W. Baumann, P. Klein, R. Linssen: Transient Calibration Process using Chip Simulation and Dynamic Modeling. 7th Conference on Design of Experiments (DoE) in Engine Development, Berlin, 2013. In [3].

[8] SystemC, Language for System-Level Modeling, Design and Verification, see http://www.systemc.org

[9] E. Chrisofakis, A. Junghanns: Faster development of automotive control software with Modelica and FMI. Workshop Design of Complex Dynamic Systems: Modelica & FMI. Dassault Systemes Campus, Paris, 19.09.2013. http://qtronic.de/doc/DaimlerQTronic_ModelicaFMI_2013.pdf

[10] A. Junghanns, J. Mauss, M. Seibt: Faster Development of AUTOSAR compliant ECUs through simulation. ERTS-2014, Toulouse.

[11] Data Model for ECU Measurement and Calibration. Programmers Guide, ASAM MCD-2 MC Version 1.6.1, released 17.02.2010. https://wiki.asam.net/display/STANDARDS/ASAM+MCD-2+MC

[12] Seeker et. al.: Scalable Multi-Core Simulation Using Parallel Dynamic Binary Translation. Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation - SAMOS'11, 2011.